## Introductory Overview Lecture

# The Deep Learning Revolution

Stanford

Christopher Manning & Russ Salakhutdinov

Stanford University ❀ Carnegie Mellon University

JSM, 2018-07-29

# Plan for the tutorial

1. Introduction to deep learning (Chris)

2. More details: model optimization, regularization, interpretation (Rus)

3. Unsupervised learning: Boltzmann machines, VAEs, GANs (Rus)

4. Extended neural net architectures and applications: convolution, recurrence, attention (Chris)

# Plan for Part I

a. The revolution: A sampling of the great things done with neural networks in the last few years

b. What is deep learning?

c. Introducing neural networks: From logistic regression to neural networks

d. Training a neural network: The backpropagation algorithm

# 1a. The revolution

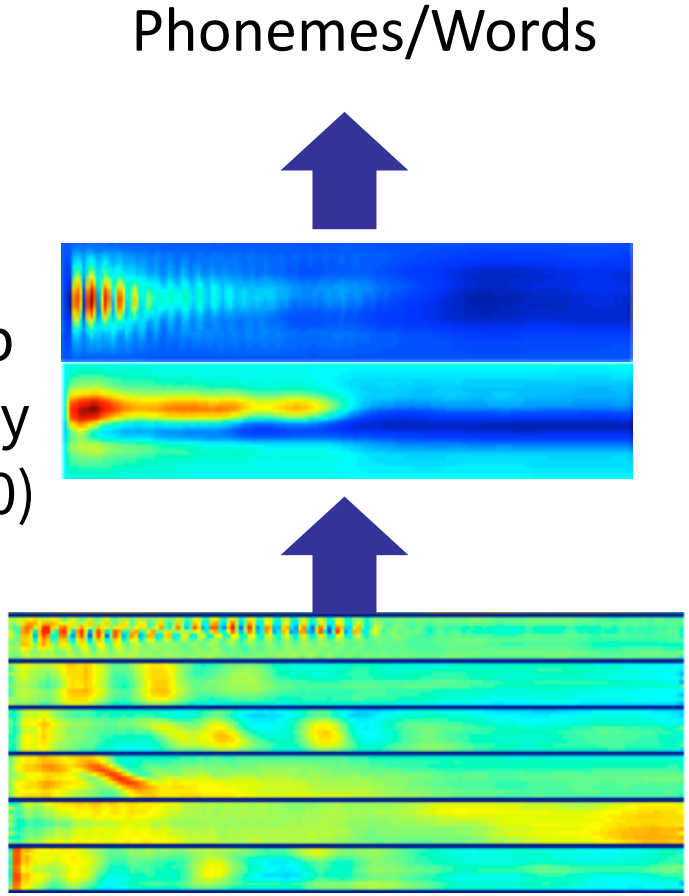
Deep learning has provided **empirically much better** methods for:

- Hard prediction problems
- Generative models of natural data distributions

Especially over high-dimensional data such as images, video, speech, text, and robotic control

# Deep Learning for Speech

- The first breakthrough results of "deep learning" on large datasets happened in speech recognition
- Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition. Dahl et al. (2010)



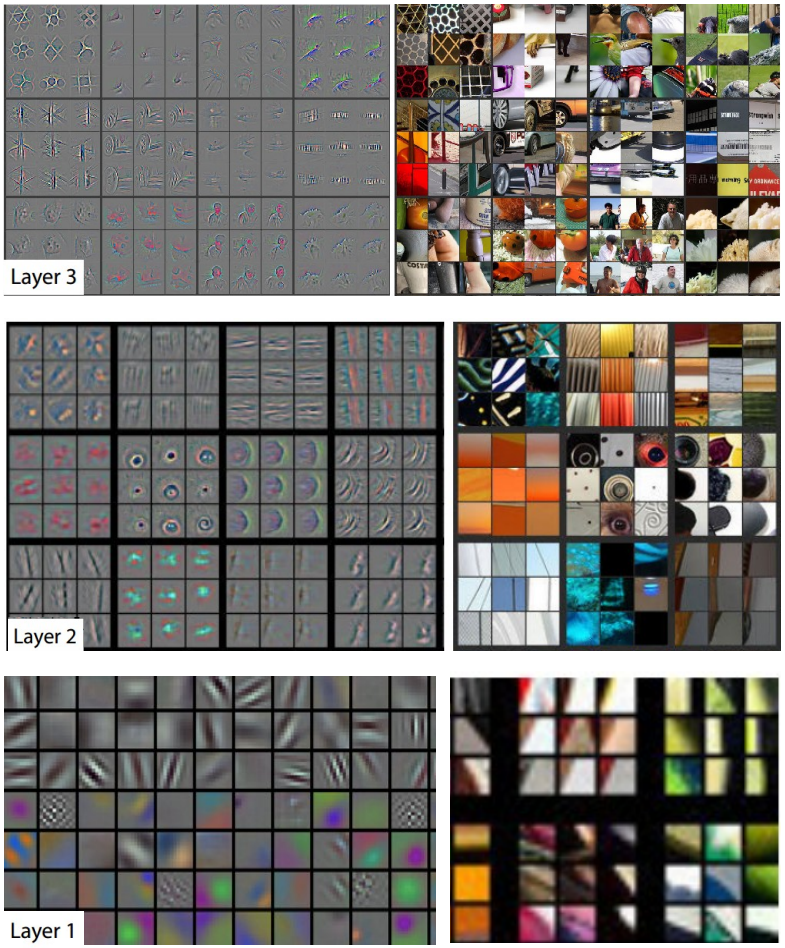

| Acoustic model Measuring WER | RT03S FSH | Hub5 SWB |
|---|---|---|
| Traditional (GMM) (2012) | **27.4** | **23.6** |
| Deep Learning (Dahl et al. 2012) | **18.5** (–33%) | **16.1** (–32%) |
| Xiong et al. (2017) | | **5.8** |

# Deep Learning for Computer Vision

First major focus of deep learning groups was computer vision

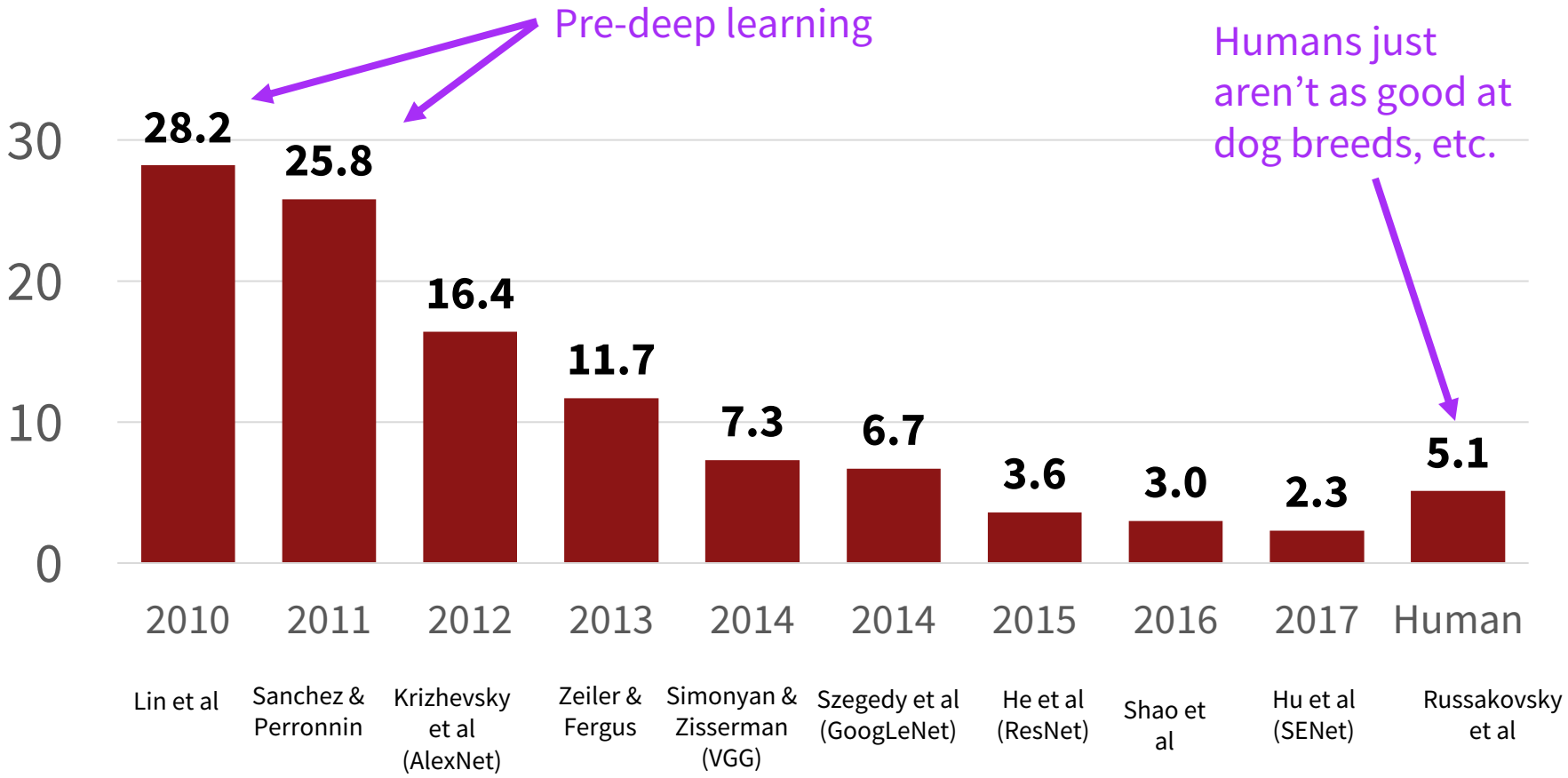
The breakthrough DL paper: ImageNet Classification with Deep Convolutional Neural Networks by Krizhevsky, Sutskever, & Hinton, 2012, U. Toronto. 37% error red.






Zeiler and Fergus (2013)

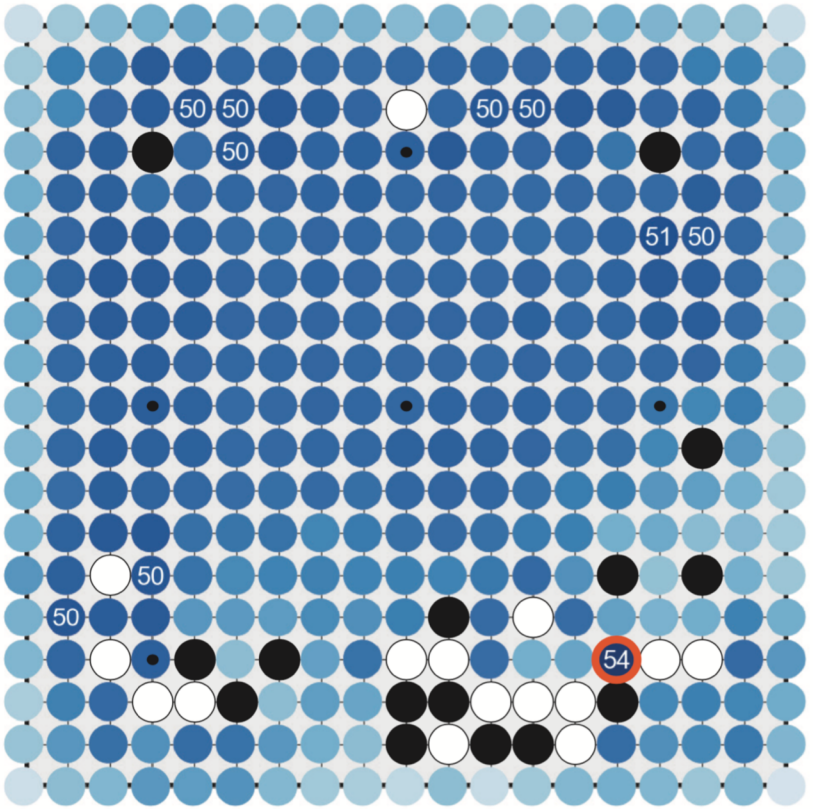
# ImageNet Classification Challenge Error rates by year



# AlphaGo:
# Next move prediction in Go

The model works from a 319-dimensional input representing the board and uses a regression model to score potential next moves

Combined with Monte Carlo Tree Search, this “solved” Go much more quickly than anyone had been imagining



# What is the current best classification method?

Here's the winning recipe for Kaggle, 2015 on:

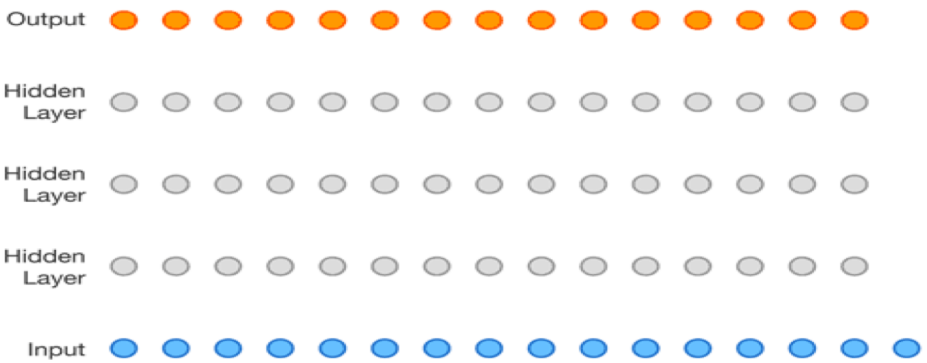
1. Careful data preprocessing, cleaning, augmentation, and feature engineering (this hasn't gone away to win Kaggle!)
2. 
    a. **For classic, structured data tables: Gradient-boosted decision trees** (xgboost). Roughly, improved MART.
    b. **For "unstructured" text, images, video, speech: Neural networks**
3. Ensembling/stacking of models, with careful cross-validation testing to find best final configuration

# Accurate modeling of the human face manifold

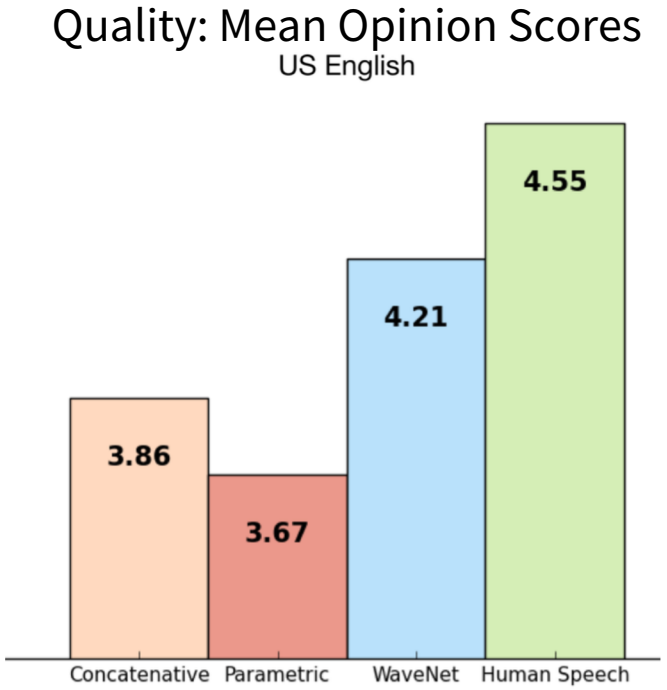
*Glow*, a reversible generative model using invertible 1x1 convolutions, learns a latent space where certain directions capture attributes like age, hair color, and so on. Kingma & Dhariwal (2018) https://arxiv.org/abs/1807.03039

# Audio generation

WaveNet: A deep generative model of raw audio
DeepMind (van den Oord et al. 2016) https://arxiv.org/abs/1609.03499




Concatenative Parametric WaveNet



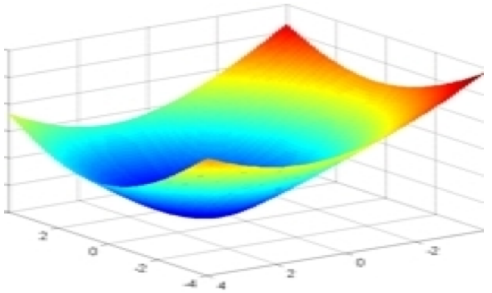
# 1b. What is Deep Learning (DL)?

- **Deep learning** is a subfield of **machine learning (statistics?)**

- Most machine learning methods work well because of **human-designed input features or representations**
  - SIFT or HOG features for vision
  - MFCC or LPC features for speech
  - Features about words parts (suffix, capitalized?) for finding person or location names

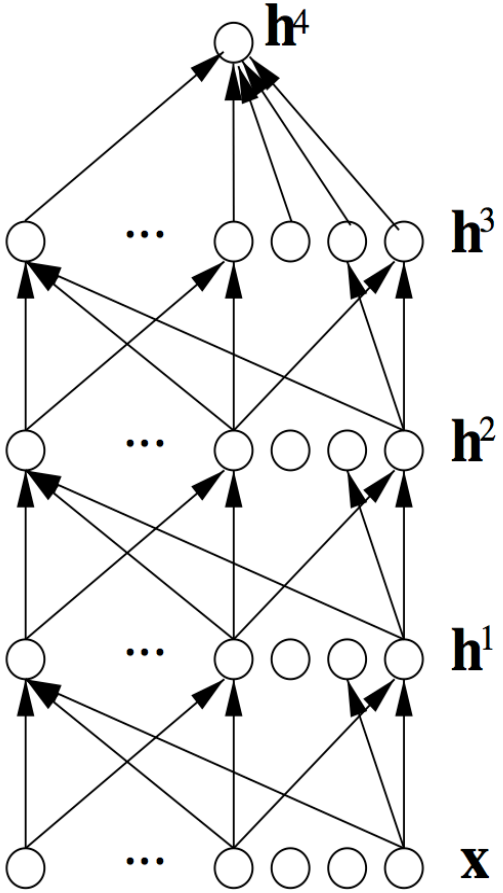
| Feature | NER |
|---|---|
| Current Word | ✓ |
| Previous Word | ✓ |
| Next Word | ✓ |
| Current Word Character n-gram | all |
| Current POS Tag | ✓ |
| Surrounding POS Tag Sequence | ✓ |
| Current Word Shape | ✓ |
| Surrounding Word Shape Sequence | ✓ |
| Presence of Word in Left Window | size 4 |
| Presence of Word in Right Window | size 4 |

- Machine learning becomes just optimizing weights to best make a final prediction



# What is Deep Learning (DL)?

- In contrast to standard machine learning,
- Representation learning attempts to automatically learn good features or representations
- Deep learning algorithms learn multiple levels of representations (here: $h^1$,$h^2$,$h^3$) and an output ($h^4$)
- From "raw" inputs **x** (e.g. sound, pixels, characters, or words)
- Neural networks are the currently successful method for deep learning
- A.k.a. "Differentiable Programming"



# Why is deep learning winning now?

- It's hard work to manually find/design good features
  - Hand-designed features are often over-specified, incomplete
  - **Learned Features** are easy to adapt, fast to learn
  - Deep learning provides a flexible framework for learning to represent information from the world
- Large amounts of training data favor deep learning
- Modern multi-core CPUs/GPUs favor deep learning
- An effective method for end-to-end system optimization
- Better context-modeling due to less independence assumptions
- Better regularization, optimization, transfer etc. methods

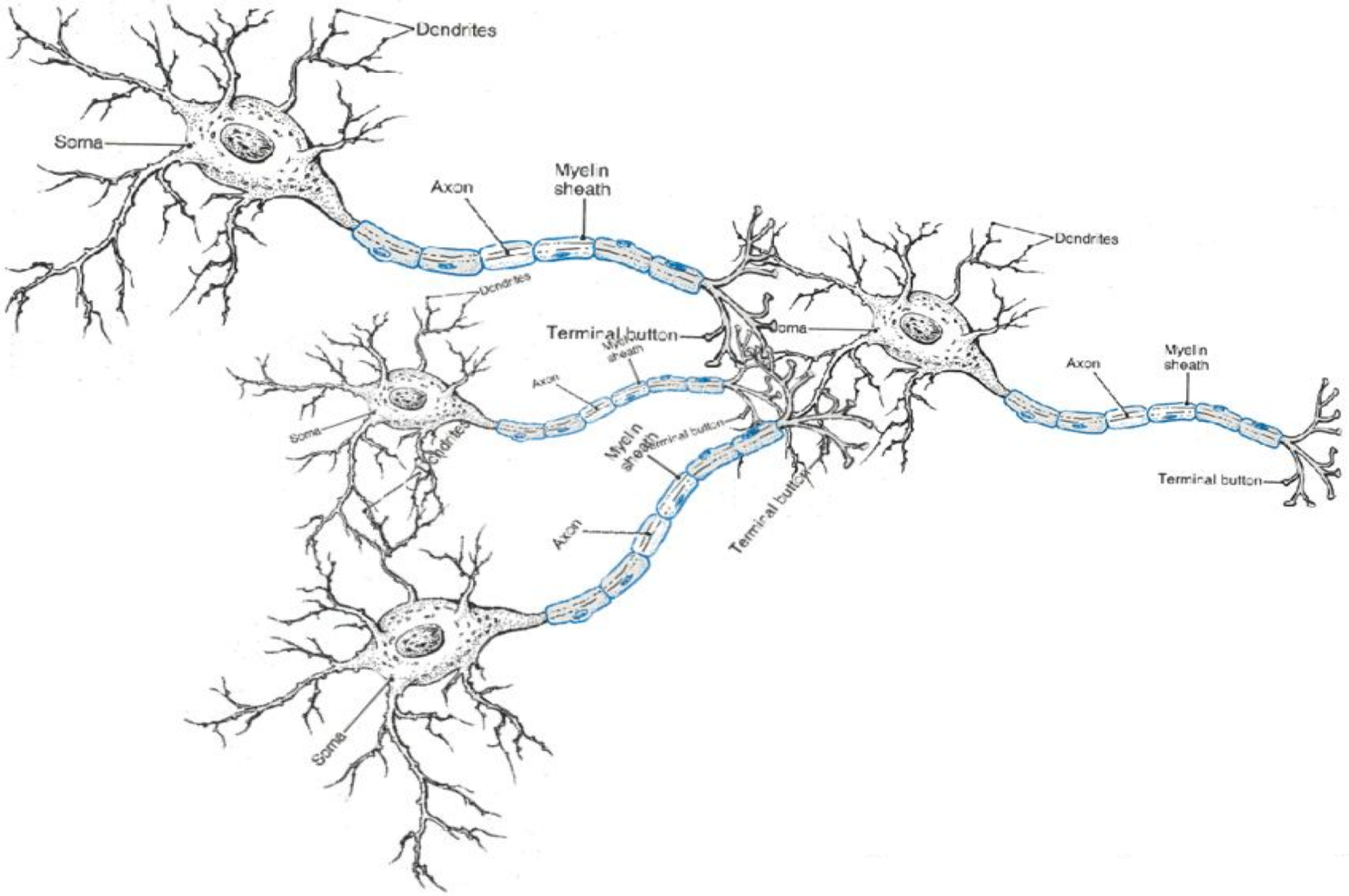
# 1c. From logistic regression to neural nets

Neural networks come with their own terminological baggage (it's their history!)

But if you understand how logistic regression models work

> Then **you already understand** the operation of a basic neuron!

# Biological neural computation

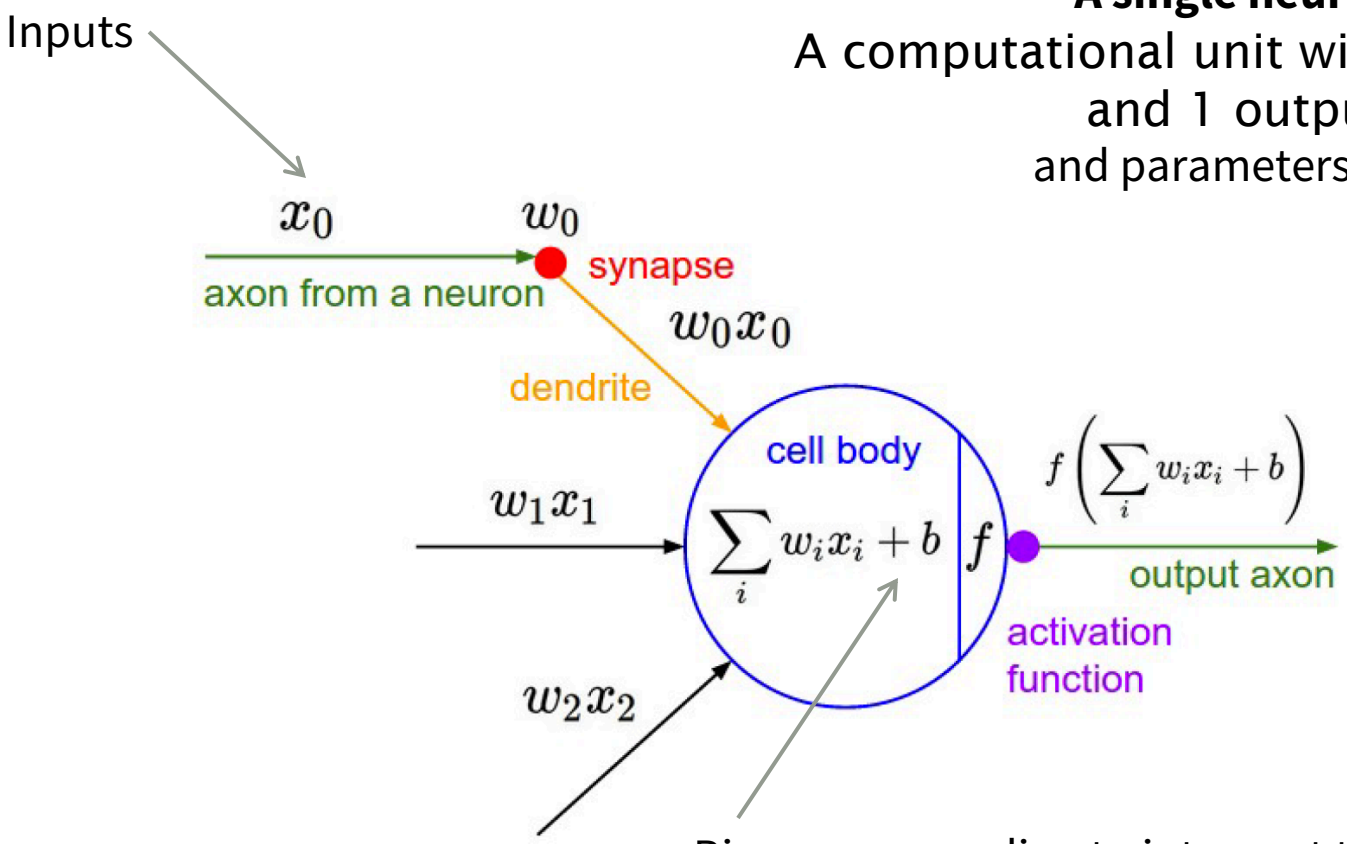


# An artificial neuron

**A single neuron**

A computational unit with *n* (3) inputs and 1 output

and parameters *w*, *b*



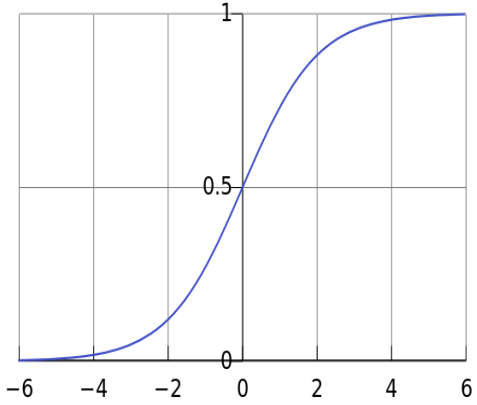
# A neuron is essentially a binary logistic regression unit

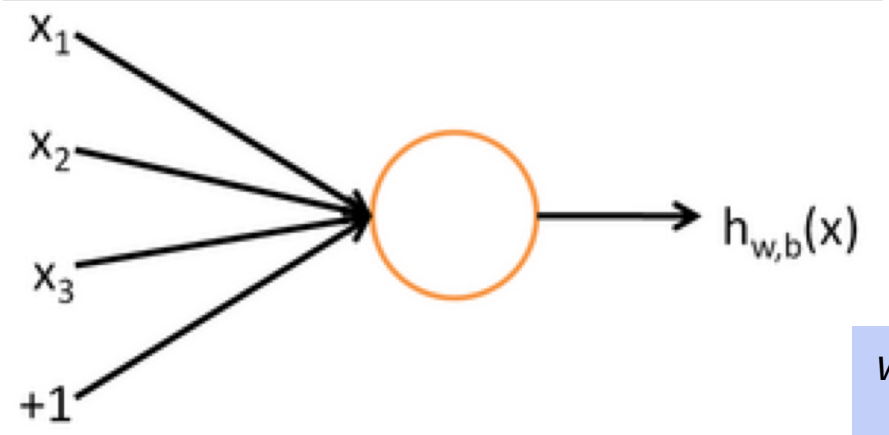
$f$ = nonlinear activation function (e.g., logistic),
$w$ = weights, $b$ = bias, $h$ = hidden, $x$ = inputs

$$h_{w,b}(x) = f(w^{\mathsf{T}}x + b)$$

*b:* We can have an "always on" feature, which gives a class prior, or separate it out, as a bias term

$$f(z) = \frac{1}{1 + e^{-z}}$$

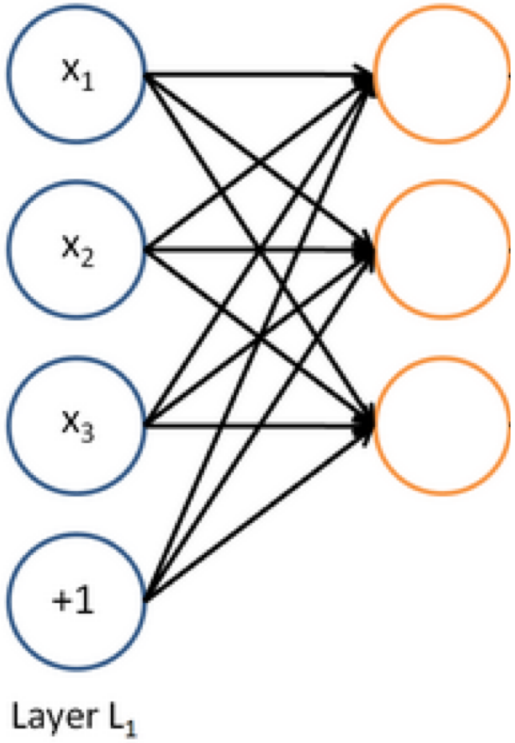



*w*, *b* are the parameters of this neuron
i.e., this logistic regression model

# A neural network = running several logistic regressions at the same time

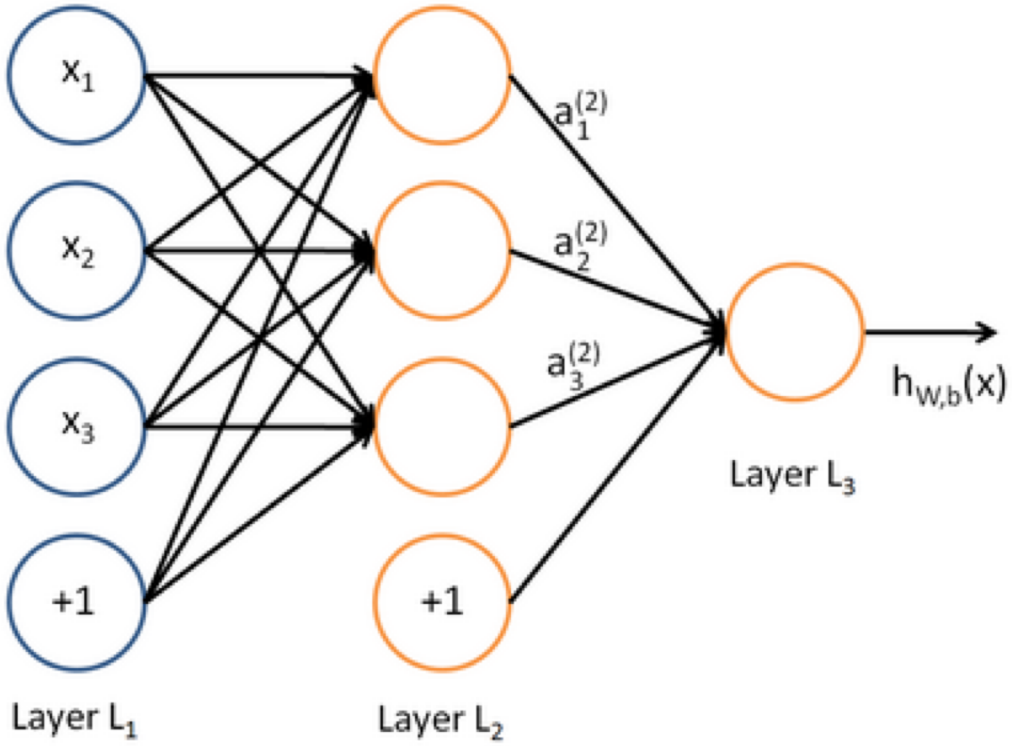
If we feed a vector of inputs through a bunch of logistic regression functions, then we get a vector of outputs …




*But we don't have to decide ahead of time what variables these logistic regressions are trying to predict!*

# A neural network = running several logistic regressions at the same time

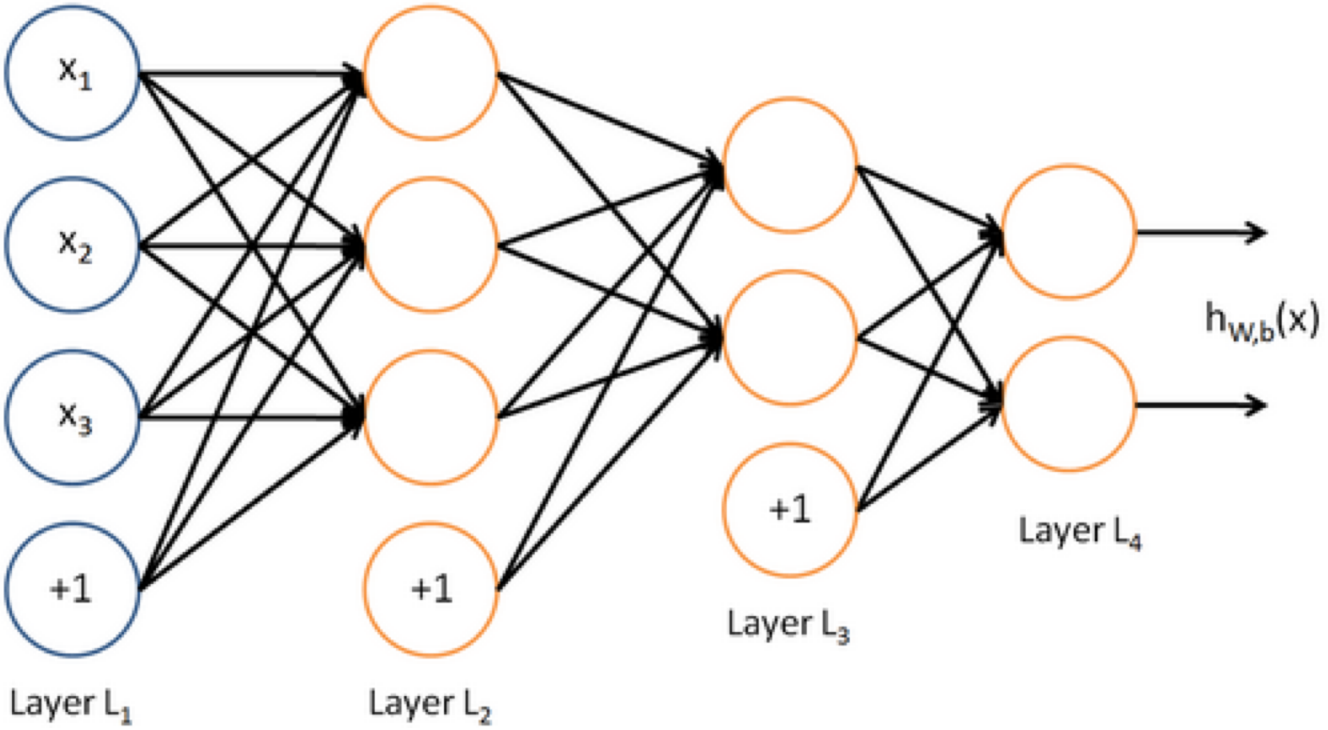
… which we can feed into another logistic regression function




*It is the top-level loss function that will direct what the intermediate hidden nodes should compute, so as to do a good job at predicting the targets for the next layer, etc.*

# A neural network = running several logistic regressions at the same time

Before we know it, we have a multilayer neural network….



# Matrix notation for a layer

We have

$$a_1 = f(W_{11}x_1 + W_{12}x_2 + W_{13}x_3 + b_1)$$

$$a_2 = f(W_{21}x_1 + W_{22}x_2 + W_{23}x_3 + b_2)$$

etc.

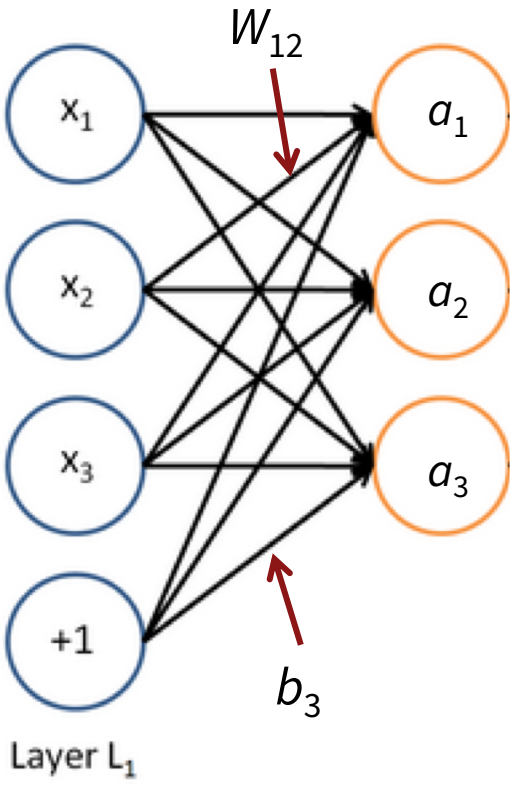
In matrix notation

$$z = Wx + b$$

$$a = f(z)$$

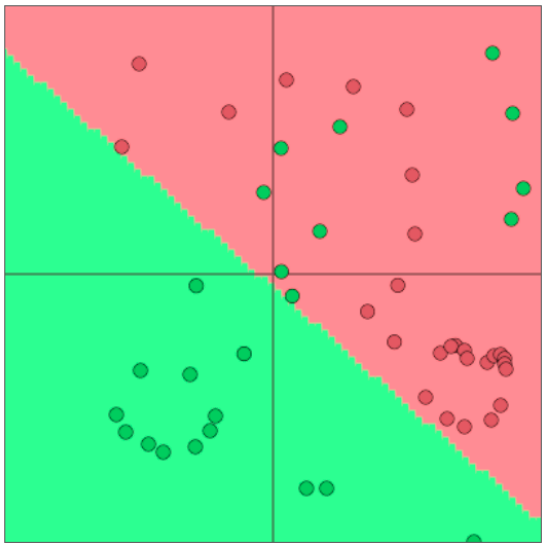
where *f* is applied element-wise:

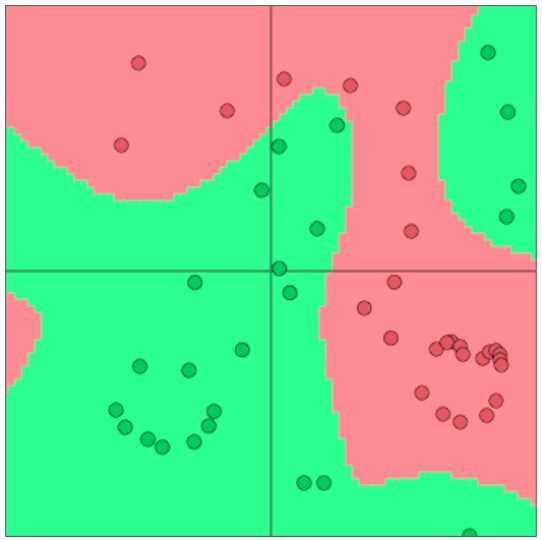
$$f([z_1, z_2, z_3]) = [f(z_1), f(z_2), f(z_3)]$$



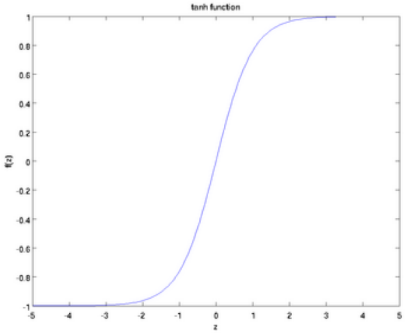
# Non-linearities (aka "f"): Why they're needed

- Without non-linearities, deep neural networks can't do anything more than a linear transform
  - Extra layers can just be compiled down: $W_1 W_2 x = Wx$
- Deep nets with non-linearities and more layers can approximate more complex functions!
- In practice, don't use logistic, but:

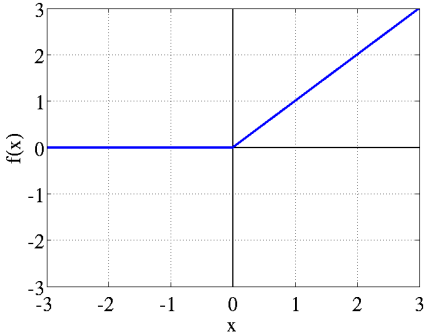




tanh




relu



# Building a supervised classifier

- We have a training dataset consisting of samples $\{x_i, y_i\}_{i=1}^{N}$
- $x_i$ are inputs, represented somehow as numeric features
  - A vector of dimension $d$
- $y_i$ are labels (one of $C$ classes) we try to predict

- Standard ML approach: assume $x_i$ are fixed, train logistic regression (or softmax regression) weights $W \in \mathbb{R}^{C \times d}$
- Predictions: For each $x$:

$$p(y|x) = \frac{\exp(W_y.x)}{\sum_{c=1}^{C} \exp(W_c.x)}$$

# Details of the softmax classifier

A softmax is itself a neural network layer

1. Take the $y^{th}$ row of $W$ and dot product that row with $x$:

$$W_{y.}x = \sum_{i=1}^{d} W_{yi}x_i = f_y$$

   Compute all $f_c$ for $c = 1,\ldots,C$

2. Apply softmax function to get normalized probability:

$$p(y|x) = \frac{\exp(f_y)}{\sum_{c=1}^{C}\exp(f_c)} = softmax(f)_y$$

# Training with softmax and cross-entropy error

- For each training example {x,y}, our objective is to maximize the probability of the correct class y

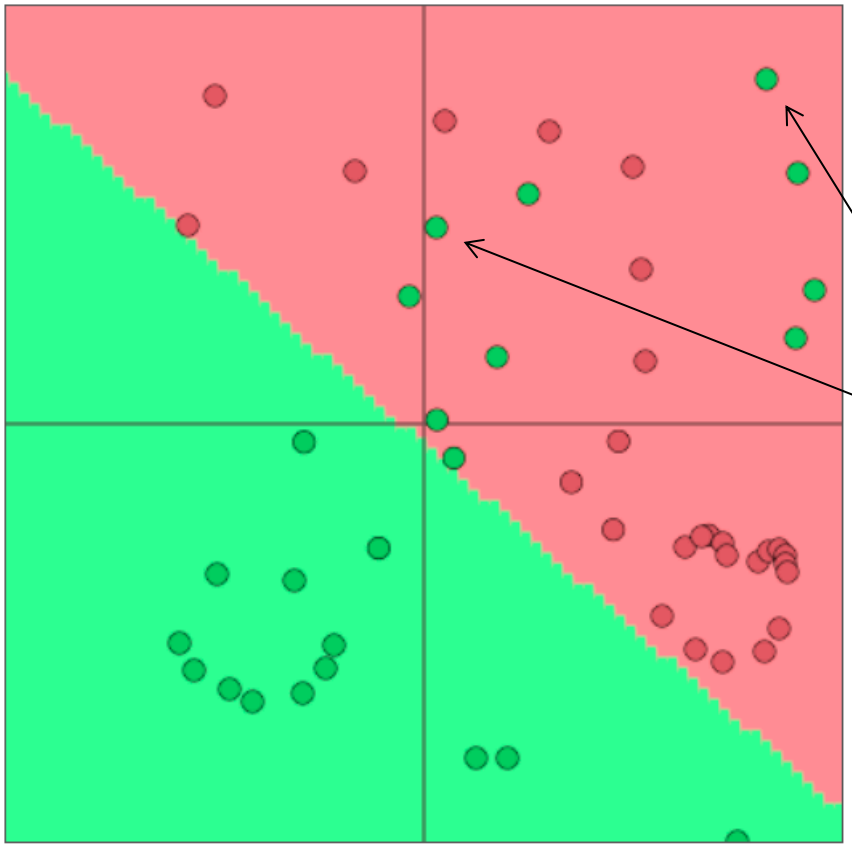
- Hence, we want to minimize the negative log probability (NLL) of that class

- This is our error or loss for the example

$$-\log p(y|x) = -\log\left(\frac{\exp(f_y)}{\sum_{c=1}^{C}\exp(f_c)}\right)$$

## Softmax (≈ logistic) regression is not very powerful

- Softmax only learns linear decision boundaries

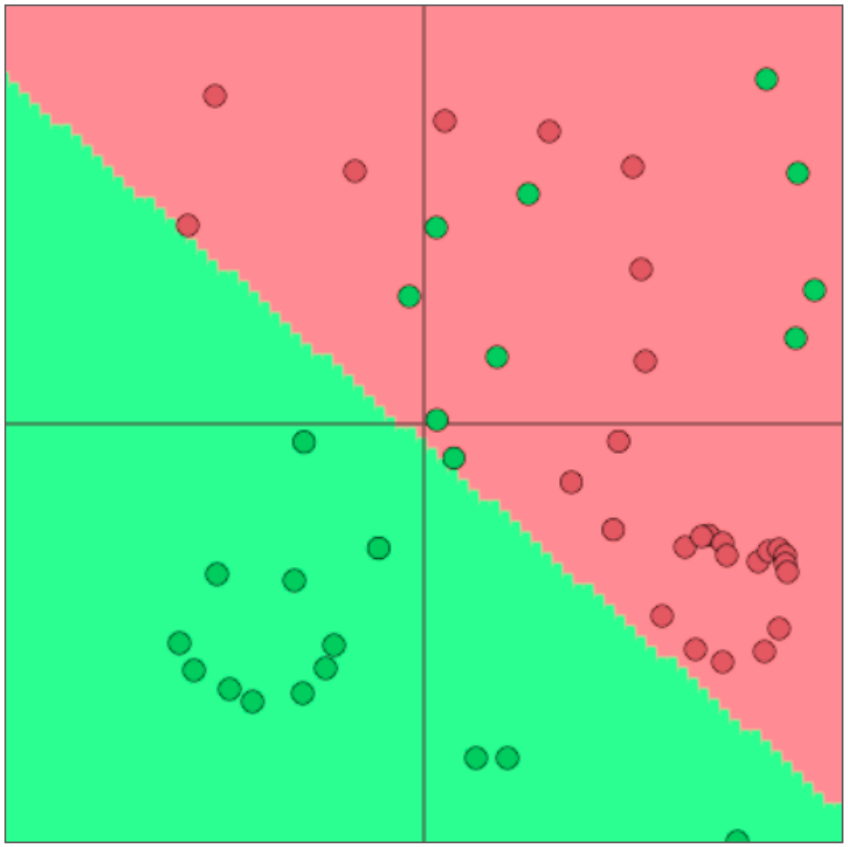


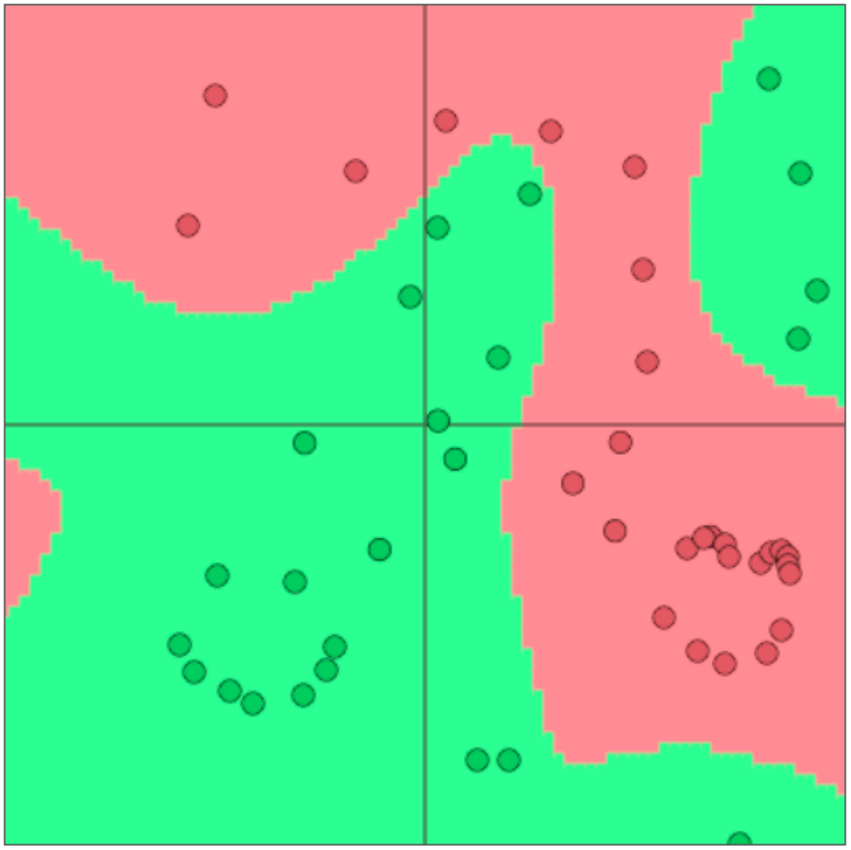
→ Unhelpful when problem is complex

It would be good to get these correct too!

# The advantages of depth

- Neural networks can learn much more complex functions and nonlinear decision boundaries!





Of course, they could have too much capacity and overfit wildly, but **the big result is:** We can train models with unbelievably many parameters and they predict **better**

# 1d. Training a neural net with Backpropagation (Rumelhart et al. 1986)

We fit a model by:

- Defining a loss on the output
- Working out gradient of example-wise loss wrt each parameter
- Adjust the parameters to shrink the loss

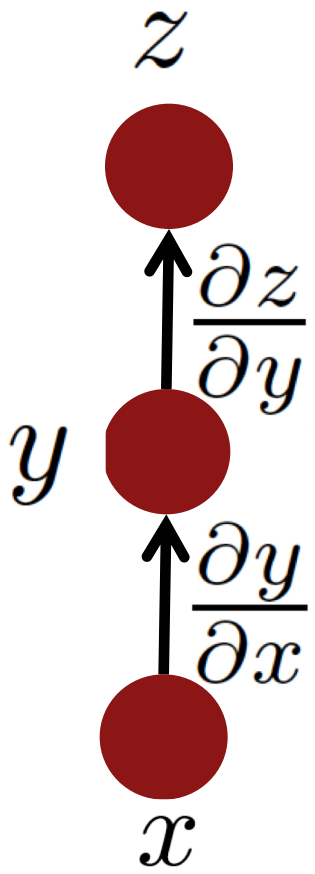
Done by "backpropagation". Backpropagation is:

- Taking derivatives and using the chain rule
- Extra trick: get efficiency by **re-using** derivatives computed for higher layers in computing derivatives for lower layers!
  - If computing the loss($x_i$, θ) is $O(n)$ then efficient gradient computation is also $O(n)$

# Simple Chain Rule

$z = f(y)$

$y = g(x)$



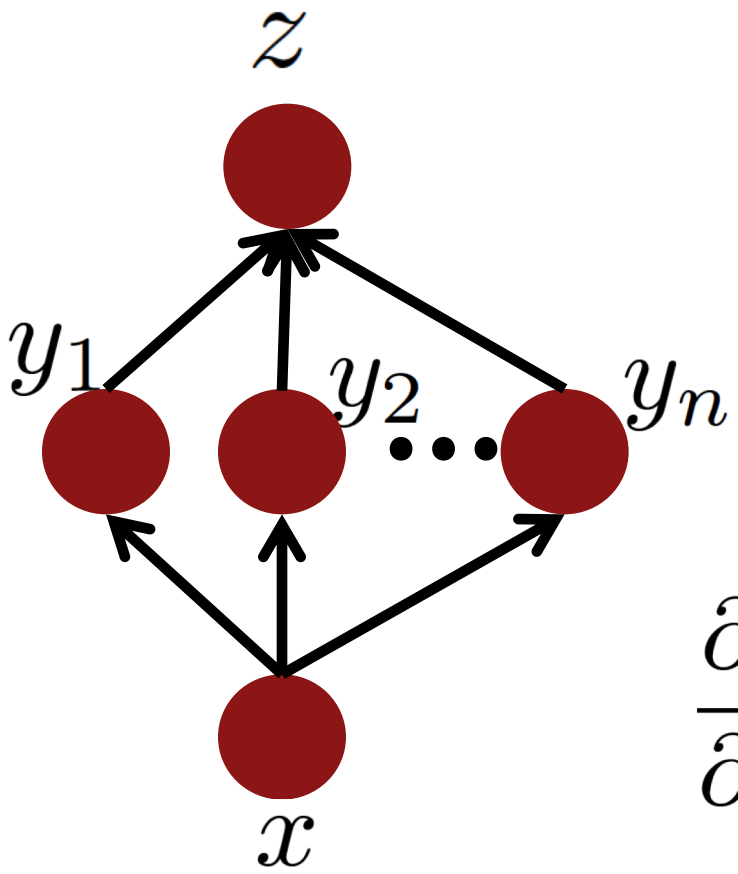

$$\Delta z = \frac{\partial z}{\partial y} \Delta y$$

$$\Delta y = \frac{\partial y}{\partial x} \Delta x$$

$$\Delta z = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \Delta x$$

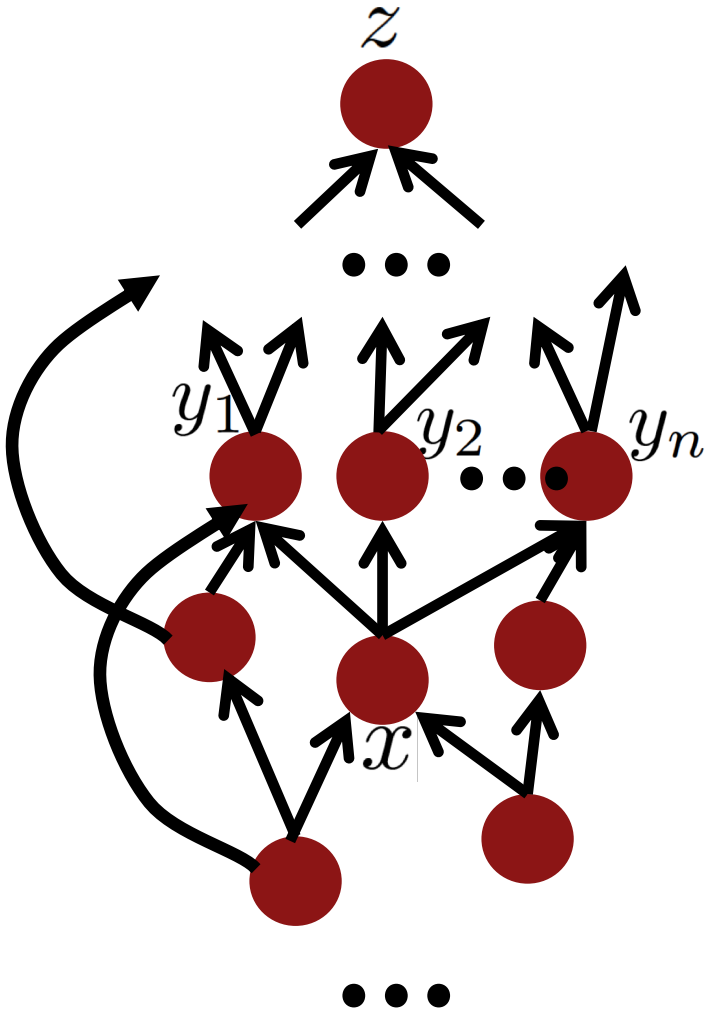
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x}$$

# Multiple Paths Chain Rule - General




$$\frac{\partial z}{\partial x} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

# Chain Rule in Computation Graph

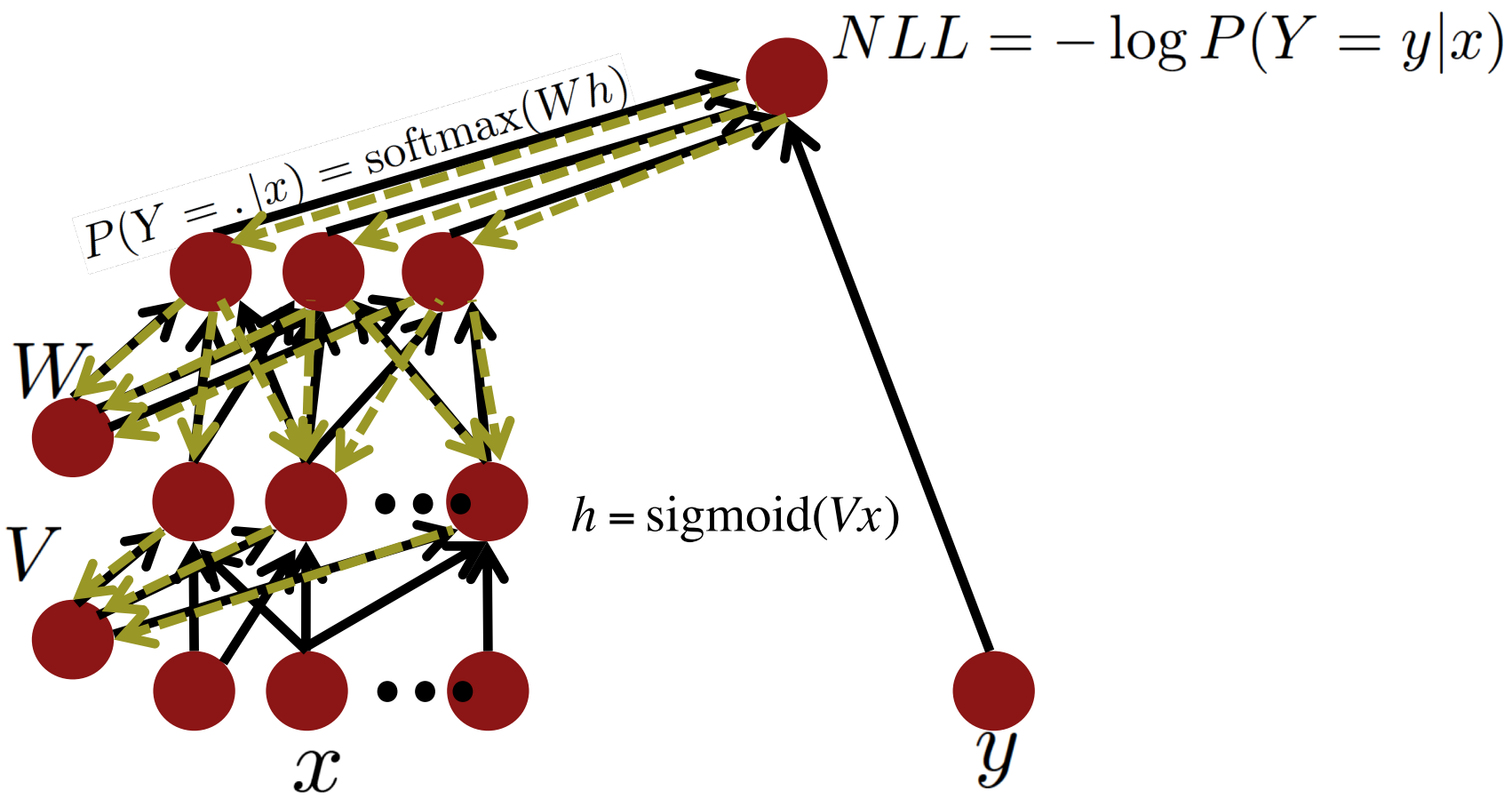



Computation graph: any directed acyclic graph
node = computation result
arc = computation dependency

$\{y_1, y_2, \ldots y_n\}$ successors of $x$

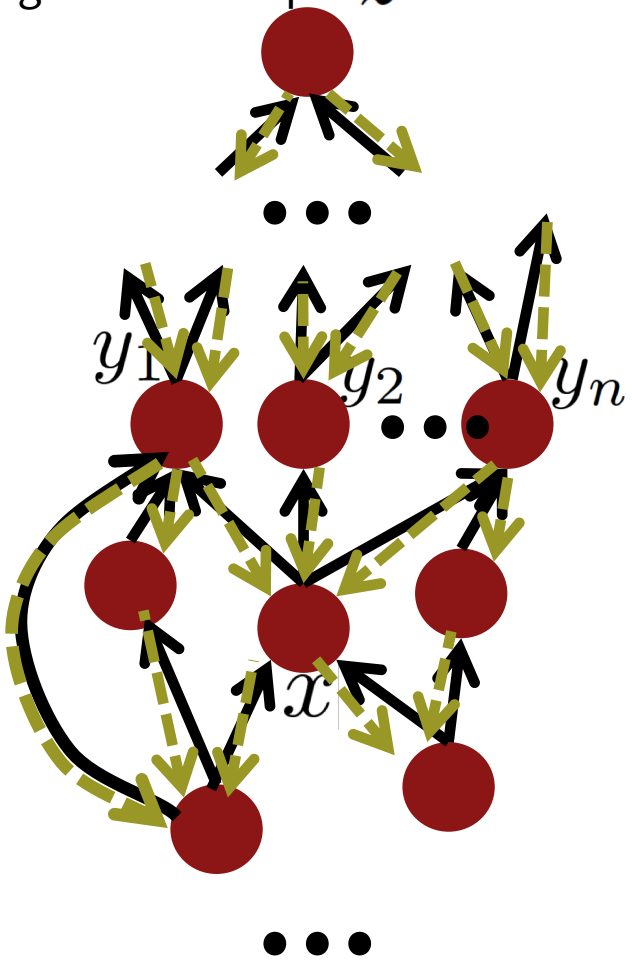
$$\frac{\partial z}{\partial x} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

# Back-Prop in Multi-Layer Net



# Back-Prop in General Computation Graph

Single scalar output $z$

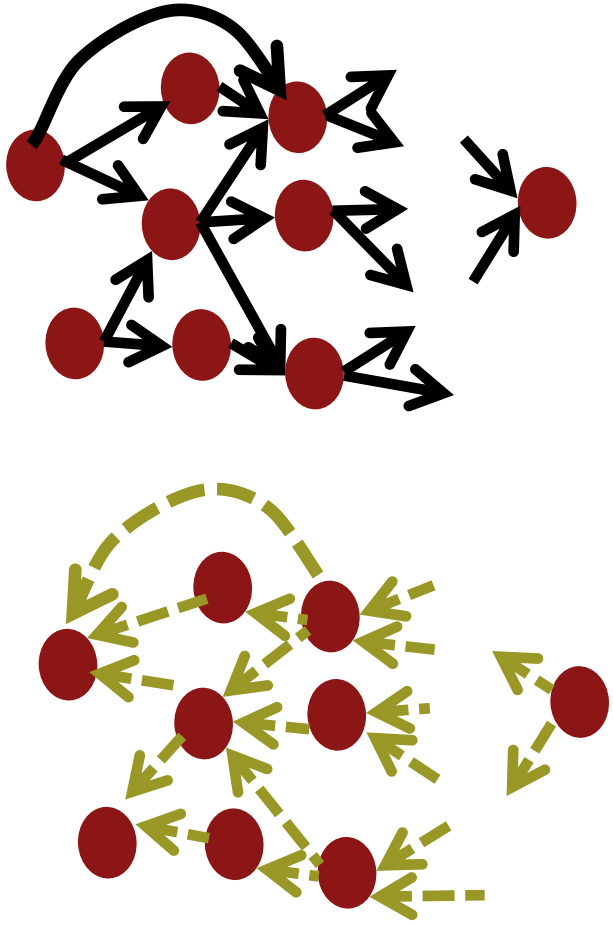



1. Fprop: visit nodes in topo-sort order
   - Compute value of node given predecessors
2. Bprop:
   - initialize output gradient = 1
   - visit nodes in reverse order:
   Compute gradient wrt each node using gradient wrt successors
   $\{y_1, y_2, \ldots y_n\}$ = successors of $x$

$$\frac{\partial z}{\partial x} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

In general out nets have regular layer-structure and so we are using matrices and Jacobians…

# Automatic Differentiation



- The gradient computation can be automatically inferred from the symbolic expression of the fprop.
- Each node type needs to know how to compute its output and how to compute the gradient wrt its inputs given the gradient wrt its output.
- Modern DL frameworks (Tensorflow, PyTorch, etc.) do backpropagation for you via automatic differentiation